Les sockets java

Introduction

- Point d'entrée entre 2 applications du réseau
- Permet l'échange de donnée entre elles à l'aide des mécanismes d'E/S (java.io)

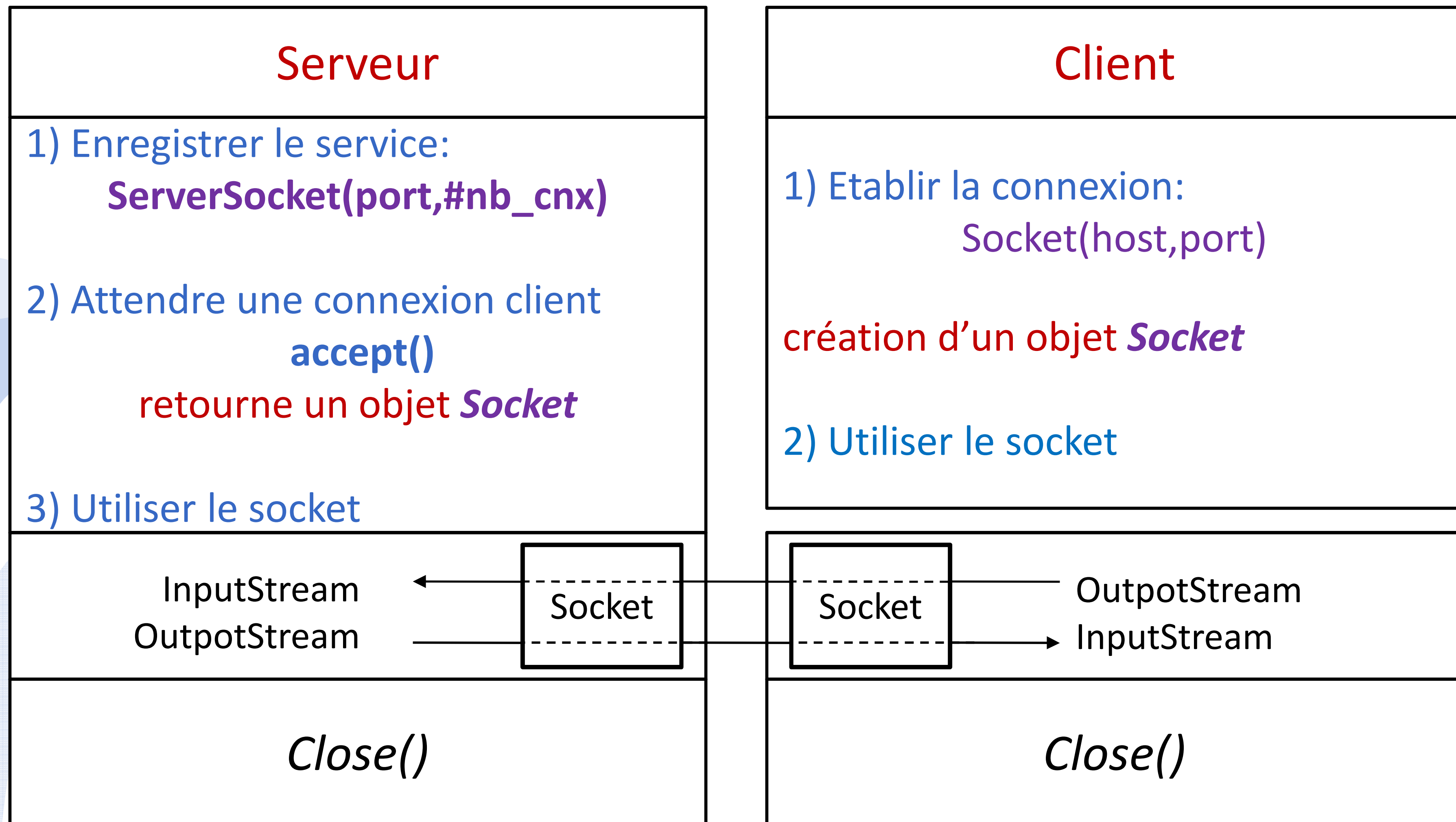
- Différents types de sockets

- Stream Sockets (TCP)
  - ✓ établir une communication en mode connecté
  - ✓ si connexion interrompue : applications informées

- Datagram Sockets (UDP)

  - ✓ établir une communication en mode non connecté
  - ✓ données envoyées sous forme de paquets
  - ✓ indépendants de toute connexion. Plus rapide, moins fiable que TCP

Le modèle C/S java



Serveur TCP

- il utilise la classe **java.net.ServerSocket** pour accepter des connexions de clients

- Quand un client se connecte à un port sur lequel un **ServerSocket** écoute,
- **ServerSocket** crée une nouvelle instance de la classe **Socket** pour supporter les communications côté serveur :

```
int port = ...;
ServerSocket server = new ServerSocket(port);
Socket connection = server.accept();   ⇐ Appel bloquant
```

java.net.ServerSocket

```
final int PORT = …;
try {
ServerSocket serveur = new ServerSocket(PORT,5);
    while (true) {
        Socket socket = serveur.accept();
    }
}
 catch (IOException e){
   ….
 }
```

Client TCP

- Le client se connecte au serveur en créant une instance de la classe **java.net.Socket** : connexion synchrone

```
String host = …;
int port = …;
Socket connection = new Socket (host,port);
```

- Le socket permet de supporter les communications côté client

- La méthode **close()** ferme (détruit) le socket

- Les constructeurs et la plupart des méthodes peuvent générer une **IOException**

- Le serveur doit être démarré avant le client. Dans le cas contraire, si le client se connecte à un serveur inexistant, une exception sera levée après un time-out

java.net.Socket

```
String HOST = "... ";
int PORT = ...;
try {
        Socket socket = new Socket (HOST,PORT);
}
try {
                socket.close(); } catch (IOException e){}
```

Les flux de données (1)

- Une fois la connexion réalisée, il faut obtenir les ***streams*** d'E/S (**java.io**) auprès de l'instance de la classe ***Socket*** en cours

- Flux entrant

✓ obtention d'un stream

```
InputSteam in = socket.getInputStream();
```

✓ création d'un stream convertissant les bytes reçus en char

```
InputSteamReader reader = new InputStreamReader(in);
```

✓ création d'un stream de lecture avec tampon: pour lire ligne par ligne dans un stream de caractères

```
BufferedReader istream = new BufferedReader(reader);
```

✓ lecture d'une chaîne de caractères

```
String line = istream.readline();
```

Les flux de données (2)

- Flux sortant

  ✓ Obtention du flot de données sortantes : bytes

  ```
  OutputSteam out = socket.getOutputStream();
  ```

  ✓ création d'un stream convertissant les bytes en chaînes de caractères

  ```
  PrintWriter ostream = new PrintWriter(out);
  ```

  ✓ envoi d'une ligne de caractères

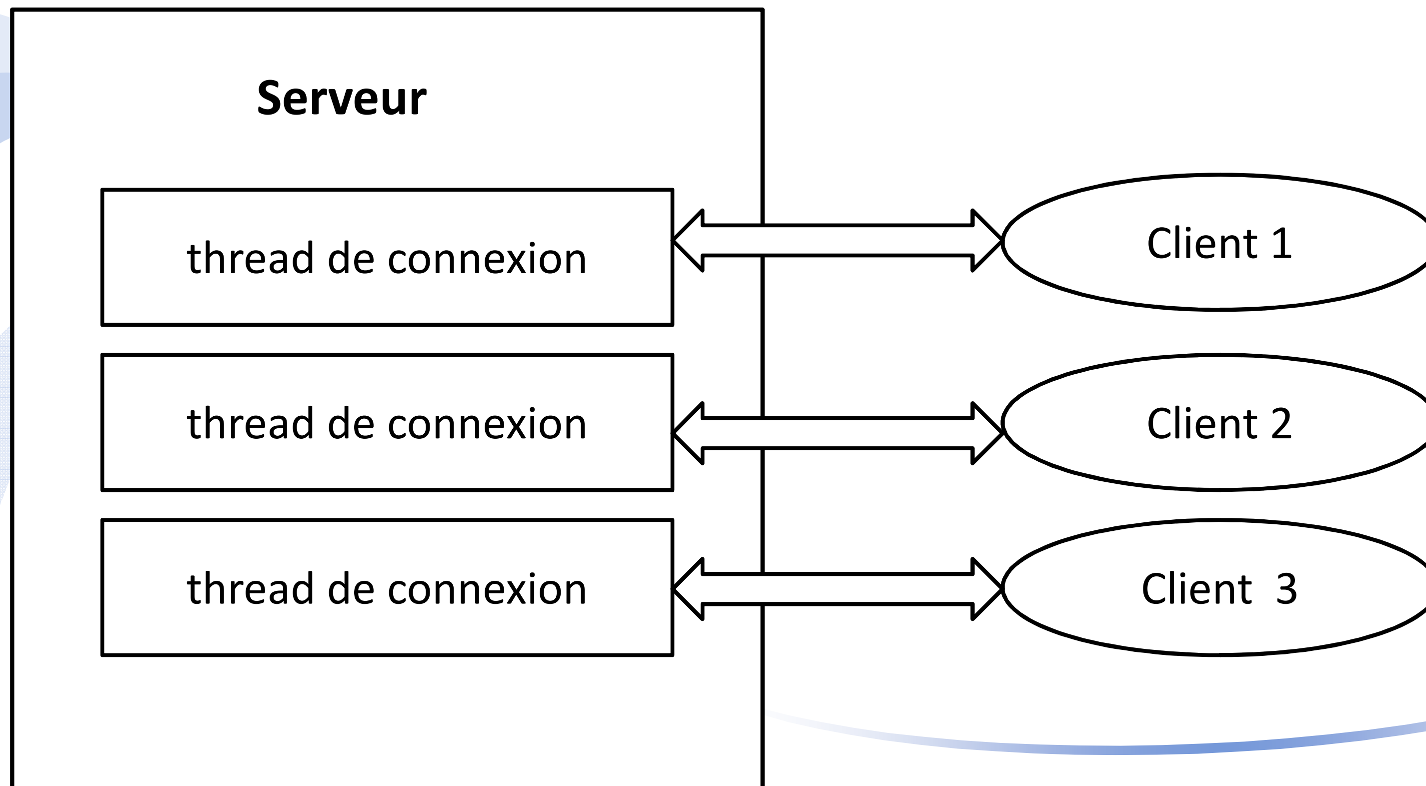
  ```
  ostream.println(str);
  ```

  ✓ envoi effectif sur le réseau des bytes (important)

  ```
  ostream.flush();
  ```

Les serveurs multiclients

- Le serveur utilise une classe **Connexion** implémentant l'interface **Runnable** (thread) pour gérer les échanges de données en tâche de fond. C'est ce thread qui réalise le service demandé.



Socket en mode datagramme

- Il faut utiliser les classes **DatagramPacket** et **DatagramSocket**
- Ces objets sont initialisés différemment selon qu'ils sont utilisés pour envoyer (***send()***) ou recevoir (***receive ()***) des paquets